Microprocessors and Microsystems 54 (2017) 1–13

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Dynamic energy management for chip multi-processors under performance constraints

Milad Ghorbani Moghaddam, Cristinel Ababei*

The Department of Electrical and Computer Engineering, Marquette University, Milwaukee WI, 53233, USA

ARTICLE INFO

Article history:
Received 10 March 2017
Revised 13 July 2017
Accepted 21 August 2017
Available online 23 August 2017

Keywords:
Chip multi-processors
Energy minimization
DVFS
Performance constraints

ABSTRACT

We introduce a novel algorithm for dynamic energy management (DEM) under performance constraints in chip multi-processors (CMPs). Using the novel concept of delayed instructions count, performance loss estimations are calculated at the end of each control period for each core. In addition, a Kalman filtering based approach is employed to predict workload in the next control period for which voltage-frequency pairs must be selected. This selection is done with a novel dynamic voltage and frequency scaling (DVFS) algorithm whose objective is to reduce energy consumption but without degrading performance beyond the user set threshold. Using our customized Sniper based CMP system simulation framework, we demonstrate the effectiveness of the proposed algorithm for a variety of benchmarks for 16 core and 64 core network-on-chip based CMP architectures. Simulation results show consistent energy savings across the board. We present our work as an investigation of the tradeoff between the achievable energy reduction via DVFS when predictions are done using the effective Kalman filter for different performance penalty thresholds.

1. Introduction

Chip multi-processors (CMPs) have become the norm in most computing systems, including desktop computers, portable devices, servers and datacenters also called warehouse scale computers (WSCs). One of the biggest challenges that designers face in this context is energy consumption, which is desired to be minimized, ideally without affecting the achievable performance. This is increasingly important due to the advent and wide spread of mobile devices but also due to the increasingly large energy consumption footprint of datacenters. We are interested in reducing energy consumption in mobile devices in order to prolong battery life. Reducing energy consumption in datacenters reduces costs and can have a beneficial impact on the environment. For example, in 2013, U.S. datacenters consumed an estimated 91 billion kilowatt-hours of electricity, enough to power twice the households in New York City. By 2020, estimated consumption will increase to 140 billion kilowatt-hours, costing American businesses $13 billion per year in electricity bills and causing the emission of nearly 150 million metric tons of carbon pollution annually [1]. According to the U.S. Energy Information Administration, that is about 7% of total commercial electric energy consumption and it is projected that this number will increase [2]. Therefore, improving energy efficiency is not only important for the cost to companies, but for the environmental footprint of these WSCs as this computing domain rapidly expands [3]. Reducing energy consumption in CMPs has the additional benefit of reducing power dissipation, which in turn lowers chip temperatures that have a beneficial impact on the lifetime reliability of these devices and systems.

One of the most popular techniques to enable energy optimizations is dynamic voltage and frequency scaling (DVFS). Lowering only the clock frequency of a core helps to reduce the average power consumption for a given application while the total energy consumption remains the same to execute the application. Reducing the average power consumption in turn lowers the chip temperature, however, at the expense of longer execution times for the application. Lowering the supply voltage helps to reduce the total power consumption and this helps in turn to reduce the total energy consumption that is needed for the execution of a given application. DVFS changes both voltage and frequency dynamically and can be used to exploit both above benefits. However, it usually comes at the price of performance degradation due to frequency throttling. In the case of multicore processors, per-core DVFS is not yet widely supported (Intel Haswell-EP and Samsung Exynos processors are known to support it). However, many recent studies have shown the benefits of per-core or per-cluser-of-cores DVFS capabilities [4–9]. Our work is under the assumption that such per-core DVFS may be possible in the future multicore processors and

* Corresponding author.
E-mail addresses: milad.ghorbanimoghaddam@marquette.edu (M.G. Moghaddam), cristinel.ababei@marquette.edu (C. Ababei).

http://dx.doi.org/10.1016/j.micpro.2017.08.005
0141-9331/

it is under this assumption that we implement and test the proposed ideas inside the Sniper system simulator.

In this paper, we introduce a novel algorithm for dynamic energy management (DEM) under performance constraints in future chip multi-processors with 16 and 64 core network-on-chip (NoC) based processors. We focus on processors where the communication is done via NoCs because they are becoming increasingly popular for processors with large numbers of cores that we investigate in our simulations. However, our ideas are applicable to traditional bus based processors as well. However, in our paper we do not investigate them because for large number of cores buses may not be scalable. The proposed algorithm uses a very effective heuristic that employs the DVFS technique and an efficient workload prediction method based on Kalman filtering. We test our algorithm extensively on a variety of benchmarks and report simulation results obtained with a system simulation tool, Sniper, that show the effectiveness of the algorithm to reduce energy consumption under performance constraints. We make our implementation publicly available so that other researchers can duplicate and compare to our results. Currently, to the best of our knowledge, we are not aware of any such publicly available algorithm implementation targeting 16 and 64 core processors which has been tested on such a large number of benchmarks.

The remainder of this paper is organized as follows. In the next section, we briefly review related literature. Then, we present background information on Kalman filtering, which is the primary estimation technique that we use in this paper. In Section 4, we present the proposed dynamic energy management of chip multiprocessors (CMPs) under performance constraints. Section 5 reports simulation results. Finally, we conclude and summarize our contributions in Section 6.

2. Related work

A large number of previous studies used dynamic voltage and frequency scaling (DVFS). DVFS is one of the most popular techniques that has been used for the optimization of various figures of merit in systems-on-chip (SoCs), chip multi-processors (CMPs), and general purpose graphics processor units (GPGPUs), including thermal profile, power, energy, and lifetime reliability. These previous studies developed optimization algorithms that use the DVFS technique to select voltage-frequency pairs for different cores or components such that the thermal profile becomes more uniform and hotspots are eliminated, power dissipation or energy consumption is reduced, and lifetime reliability is prolonged with minimal degradation in performance.

Previous works have employed a variety of methods including machine learning [10–12], game theory [13], and convex optimization [14] to find the optimal voltage-frequency pairs to manage the energy consumption of homogeneous (i.e., formed by identical cores) processor. More recently, heterogeneous processors are exploited towards additional optimization opportunities. For example, the study in [15] proposed a joint temperature and energy management solution for heterogeneous multicore processors. Their heuristic uses both DVFS and temperature- and performance-aware task assignment strategy that maximizes the energy savings, while maintaining the temperature at safe levels.

While in this paper we focus at the processor level and assume per-core DVFS capability, previous work employed DVFS also at the cluster of compute nodes levels. For example, the study in [17] presented a DVFS that automatically adapted the voltage and frequency for energy savings at runtime in high performance computing clusters formed by four Athlon64-based compute nodes connected via Gigabit Ethernet and another four-node quad-CPU cluster based on the Celestica A8440 server. Similarly, the study in [18] presented a performance-prediction model that is used by a per-CPU DVFS algorithm that makes DVFS decisions based on the index of CPU intensiveness. They verified the algorithm in a 9-node power-aware cluster formed by dual core processors. Other DVFS algorithms applied at the cluster of CPU nodes level include [19,20]. Some recent work also took a more holistic approach and applied DVFS to both CPU and the DRAM subsystem to achieve additional energy savings. They reported for a server platform with an Intel i5-4590 quad-core processor and 8 GB of main memory as much as 22% energy savings with a low performance loss of only 4.8%.

Most previous works focusing at the processor level rely on some form of performance estimates, which are used inside such optimization algorithms. These estimates are computed usually by various counters that count stall cycles and last-level (e.g., L2) cache misses [23], leading loads cycles (due to loads as non-speculative reads that result in last-level cache misses) [24], enhanced leading loads cycles (takes into account both variable memory access latency and performance effects of prefetching) [25], and off-chip (L2) I-cache misses and off-chip (L2) D-cache load misses [26].

The majority of these methods focus on last-level cache miss cycles and non-pipelined stall cycles, thus, they are very sensitive to the accuracy of counting misses and stalls. In this paper, we eliminate the issues related to counting these misses and stalls by proposing a new technique called delayed instruction count. We present a new heuristic algorithm for dynamic energy management (DEM) of chip multi-processors. It also employs the DVFS technique to identify the best VF pairs for all cores of the CMP. This is done using accurate and efficient estimations of the average cycles per instruction and the instruction count, which are done using a Kalman filtering technique. Our algorithm uses the Kalman filter technique, which has been proven to provide accurate estimations. For example, recently, the study in [16] employed a combination of recursive least squares (RLS) and Kalman filters (KF) to estimate processor package temperature and to construct a dynamic energy management controller to predict the optimal voltage/frequency setting to achieve maximum energy efficiency under temperature constraints. The following highlights the main contributions of this paper:

- The proposed energy management algorithm is very efficient and effective. It provides consistent energy savings under a given performance constraint for all benchmarks that we investigated.
- Because the proposed algorithm is implemented for each core separately it is easily scalable even to heterogeneous multiprocessors and systems as well.
- We investigate future network-on-chip based chip multiprocessors with 16 and 64 core architectures.

We would like to emphasize that our work is an investigation of the tradeoff between the achievable energy reductions via DVFS when predictions are done using the effective Kalman filter for different performance penalty thresholds rather than a claim of the best dynamic energy management approach out there, which is actually challenging to identify. That is because there are many previous studies who proposed various energy reduction techniques and who made various claims for single or multicore processors, for bus based or network-on-chip based multicores, with validations in simulation or on real hardware but for relatively small number of cores. Please note that none of the previous approaches made their implementation publicly available. This makes replication of results very difficult if not impossible.

3. Background on Kalman filtering

In this section, we present a brief description of Kalman filtering, which we use later in this paper as the primary estimation technique to estimate the average cycles per instruction and the instruction count for the next control period. The Kalman filter uses a set of recursive equations and employs a feedback control mechanism in a way that minimizes the variance of the estimation error [27]. It is an adaptive filter applied to predict the state x of a discrete-time controlled process. Using the notation from [28], the process can be described by the following state and output equations.

$$x_n = Ax_{n-1} + Bu_{n-1} + w_{n-1} \tag{1}$$

$$z_n = Hx_n + v_n \tag{2}$$

Where A, B, and H are matrices usually. A is the state transition model applied to the previous state x_{n-1}. It relates the states at time steps $n-1$ and n, in the absence of process noise or control input. B relates the optional control input u to the state x, and the matrix H relates the state x to the measurement or observation z. The random variable w_{n-1} models the process noise assumed to be a white Gaussian noise with zero mean and covariance Q, $w \sim N(0, Q)$. Similarly, the random variable v_n is the measurement noise also assumed to have a Gaussian distribution with zero mean and covariance R, that is independent from Q, $v \sim N(0, R)$.

Then, we define the *a priori* and *a posteriori* estimate errors as $e_{\bar{n}} = x_n - \hat{x}_n^-$ and $e_n = x_n - \hat{x}_n$, where $\hat{x}_n^-$ is our a priori state estimate given the knowledge on the process prior to step n and $\hat{x}_n$ is our a posteriori state estimate after measurement z_n has been made. Based upon these estimates, the a priori and a posteriori estimate error covariances are given by the following expressions:

$$P_n^- = E[e_n^- e_n^{-T}] \tag{3}$$

$$P_n = E[e_n e_n^T] \tag{4}$$

To estimate the states of a process with measurements, the Kalman filter employs a feedback control technique in which the state at some time is estimated first and feedback is then provided in the form of noisy measurements. Thus, a Kalman filter is constructed in two phases. The first phase is called the *predict phase* (also called the time update phase), and here the state x is predicted *a priori* as $\hat{x}_n^-$. The second phase is called the *update phase* (also called the measurement update phase). This is where the predicted $\hat{x}_n^-$ is updated *a posteriori* as $\hat{x}_n$.

In the predict phase, the filter first projects the state ahead from the previous state $\hat{x}_{n-1}$ and certain input matrix Bu_{n1}. The filter then projects the error covariance ahead with process noise covariance Q. The two equations that accomplish that are:

$$\hat{x}_n^- = A\hat{x}_{n-1} + Bu_{n-1} \tag{5}$$

$$P_n^- = AP_{n-1}A^T + Q \tag{6}$$

Where P_n^- and P_n represent the estimated error covariance for a priori and a posteriori errors, respectively, at time n. They are calculated as shown in Eqs. (3) and (4).

The update phase starts right after the predict phase with the measurement of the actual state value at time n. The three equations utilized in this phase are:

$$K_n = P_n^- H^T (HP_n^- H^T + R)^{-1} \tag{7}$$

$$\hat{x}_n = \hat{x}_n^- + K_n(z_n - H\hat{x}_n^-) \tag{8}$$

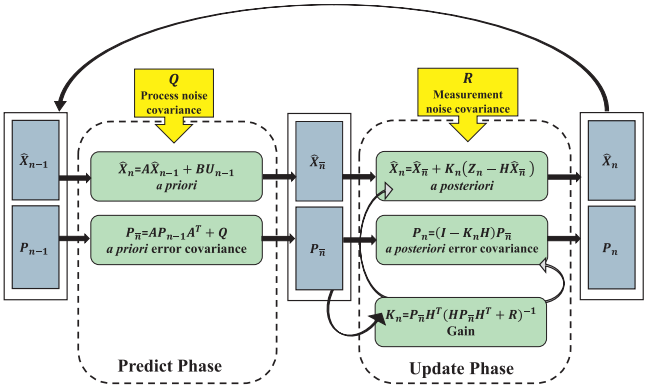


Fig. 1. Kalman filter predict phase and update phase procedure.

$$P_n = (1 - K_nH)P_n^- \tag{9}$$

The Kalman gain, K_n, is first computed by using the a priori estimate error covariance P_n^- and measurement noise covariance R. It is chosen to maximize the a posteriori error covariance P_n. The filter then updates the current state matrix $\hat{x}_n$ and a posteriori estimate error covariance P_n, using the Kalman gain. Fig. 1 shows how Kalman filter works.

In this paper, we have selected the use the Kalman filter based on our investigations and literature survey that we conducted before. We have found that the Kalman filter had been proven to be one of the best techniques in terms of complexity of implementation in software only, efficiency, and effectiveness in making accurate predictions over a short horizon. Because, in this paper, we are interested in investigating the tradeoff between energy savings and performance degradation, for varying performance degradation thresholds, we believe that the Kalman filter technique is an effective way to study this tradeoff.

4. Dynamic energy management under performance constraints

In this section, we describe the proposed DVFS based dynamic energy minimization (DEM) algorithm under performance constraints. We first describe how prediction is employed in order to proactively estimate performance losses and then we discuss the block diagram of the algorithm, which will be implemented on top of a CMP simulator and used later to conduct simulation experiments.

4.1. Performance loss estimation

The idea of the proposed dynamic energy management is to continuously monitor the CMP system operation and to periodically make decisions to *tune* different control *knobs* with the objective of shifting the system's operation to states where energy consumption is reduced as much as possible but without degradation of performance beyond the user specified threshold. In our case the *knob* is represented by the voltage-frequency (VF) pairs, which can be set individually for each of the cores of the CMP processor, effectively done as part of the DVFS algorithm. This is under the assumption that by default (i.e., in the reference or base case) all cores operate at the highest frequency to achieve the best possible performance. *Tuning* the knob translates into frequency throttling or frequency increase (if throttling has been done before for a given core), at opportune times, in order to save energy.

The key challenge in achieving that is to find a way to dynamically change between frequency voltage pairs such that the performance degradation is not more than the acceptable threshold,

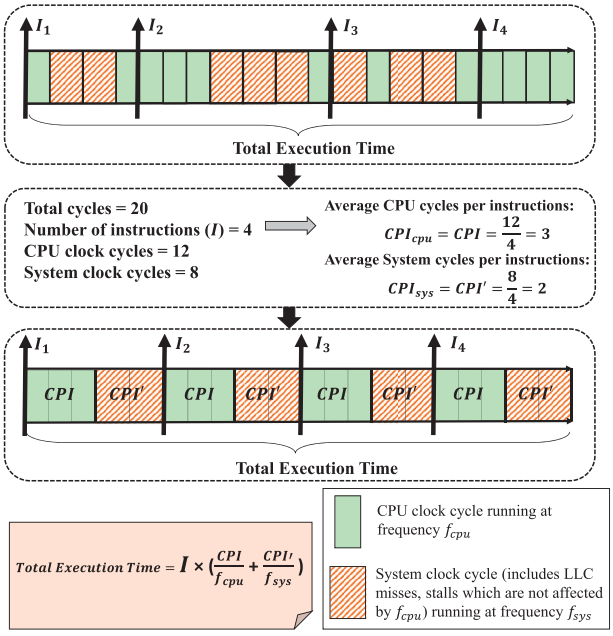


Fig. 2. Example utilized to illustrate the two different average cycles per instruction, *CPI* and *CPI′*, which are used to estimate the total execution time.

which is set by the user as a percentage, such as 5% performance degradation. The performance constraint is what complicates the problem in this case. We address this challenge by introducing a new concept, that of *delayed instructions count (DIC)*, which we use to calculate dynamically the amount of *performance loss (PL)* that we would introduce if we were to switch the current voltage-frequency pair for a given core for the *next control period* to a throttling pair (i.e., lower frequency). This amount of performance loss is estimated with respect to the reference case, which is always that of the highest frequency. We describe next the derivation of the expression that we propose to use for the estimation of performance loss.

Assume that we denote with *CPI* (cycles per instruction) the average CPU cycles per instruction when the CPU is not stalled and does useful work at a clock frequency that we refer to as f_{cpu}. Assume also that we denote with *CPI′* the average system cycles per instruction that the CPU is stalled because it has to wait due to branch misses, TLB misses, lowest level cache misses, pipeline stalls, etc. Note that with these notations, we consider the execution of a given instruction as being made up of two portions. One portion is given by the *CPI* as average number of CPU frequency cycles per instruction and the other portion is given by *CPI′* as average number of system frequency cycles per instruction. This is illustrated for a simple example in Fig. 2.

The performance of a processor for a given application is quantified via the total execution time, T_{total}, given by the following expression.

$$T_{total} = I \times \left(\frac{CPI}{f_{cpu}} + \frac{CPI'}{f_{sys}} \right) \tag{10}$$

Where I is the total number of instructions, f_{cpu} is the clock frequency that the processor is operated at, and f_{sys} is the system clock frequency.

The total execution time of the application is partitioned into a number of control periods. During the execution of the application, the system calls the proposed algorithm at the end of each control period in order to decide about the VF pairs for all cores during the next control period. Assume we refer to such periods with the generic index P. Then, applying the same rationale as that for deriving Eq. (10), for a generic control period P, we can write the expression for the duration of the period T_P as:

$$T_P = I_{P_{Done}} \times \left(\frac{CPI_P}{f_P} + \frac{CPI'_P}{f_{sys}} \right) \tag{11}$$

Where, CPI_P and CPI'_P are the average cycles per instruction during period P. $I_{P_{done}}$ represents the number of instructions executed during period P when the core operates at a particular clock frequency f_P.

Similarly, having the same average cycles per instruction, if the execution is done at the highest frequency f_H, a control period of the same walltime duration would have executed say $I_{P_{Max}}$ instructions.

$$T_P = I_{P_{max}} \times \left(\frac{CPI_P}{f_H} + \frac{CPI'_P}{f_{sys}} \right) \tag{12}$$

Using the Eqs. (11) and (12), the expression for $I_{P_{max}}$ can be derived as:

$$I_{P_{max}} = I_{P_{Done}} \times \left(\frac{\frac{CPI_P}{f_P} + \frac{CPI'_P}{f_{sys}}}{\frac{CPI_P}{f_H} + \frac{CPI'_P}{f_{sys}}} \right) \tag{13}$$

Which can be rewritten as:

$$I_{P_{max}} = I_{P_{Done}} \times \left(\frac{CPI_P(\frac{f_H}{f_P}) + CPI'_P(\frac{f_H}{f_{sys}})}{CPI_P + CPI'_P(\frac{f_H}{f_{sys}})} \right) \tag{14}$$

Let us denote as $I_{P_{NotDone}} = I_{P_{max}} - I_{P_{Done}}$ the number of instructions that turned out not to be executed or done in the last control period due to the fact that the frequency was throttled from f_H to f_P. These delayed or postponed instructions will introduce a delay penalty compared to the case when all the instructions would have been run at f_H. This number of instructions is what we call the *delayed instructions count (DIC)*, which are postponed for later, and which will result in some performance degradation. The expression for it can be written as:

$$I_{P_{NotDone}} = I_{P_{Done}} \times \left(\left(\frac{CPI_P(\frac{f_H}{f_P}) + CPI'_P(\frac{f_H}{f_{sys}})}{CPI_P + CPI'_P(\frac{f_H}{f_{sys}})} \right) - 1 \right) \tag{15}$$

That can be simplified to:

$$I_{P_{NotDone}} = I_{P_{Done}} \times \left(\frac{CPI_P(\frac{f_H}{f_P} - 1)}{CPI_P + CPI'_P(\frac{f_H}{f_{sys}})} \right) \tag{16}$$

Using again an expression similar to that from Eq. (10), we can calculate the extra time it would take to run $I_{P_{NotDone}}$ instructions at the highest clock frequency f_H. Let us refer to that as $T_{P_{delay}}$, which is essentially the penalty incurred in control period P because of running at a lower frequency, f_P:

$$\begin{aligned} T_{P_{delay}} &= I_{P_{NotDone}} \times \left(\frac{CPI_P}{f_H} + \frac{CPI'_P}{f_{sys}} \right) \\ &= \left(I_{P_{Done}} \times \left(\frac{CPI_P(\frac{f_H}{f_P} - 1)}{CPI_P + CPI'_P(\frac{f_H}{f_{sys}})} \right) \right) \times \left(\frac{CPI_P}{f_H} + \frac{CPI'_P}{f_{sys}} \right) \end{aligned} \tag{17}$$

Which can be reduced to:

$$T_{P_{delay}} = I_{P_{Done}} \times \left(\frac{CPI_P(\frac{f_H}{f_P} - 1)}{f_H} \right) \tag{18}$$

Finally, the total performance loss (PL) that is incurred over all the control periods, is calculated as the following summation:

$$PL = \sum_{P=1}^{N} \frac{T_{P_{delay}}}{T} = \sum_{P=1}^{N} \frac{I_{P_{Done}} \times \left(\frac{CPI_P(\frac{f_H}{f_P} - 1)}{f_H} \right)}{T} \tag{19}$$

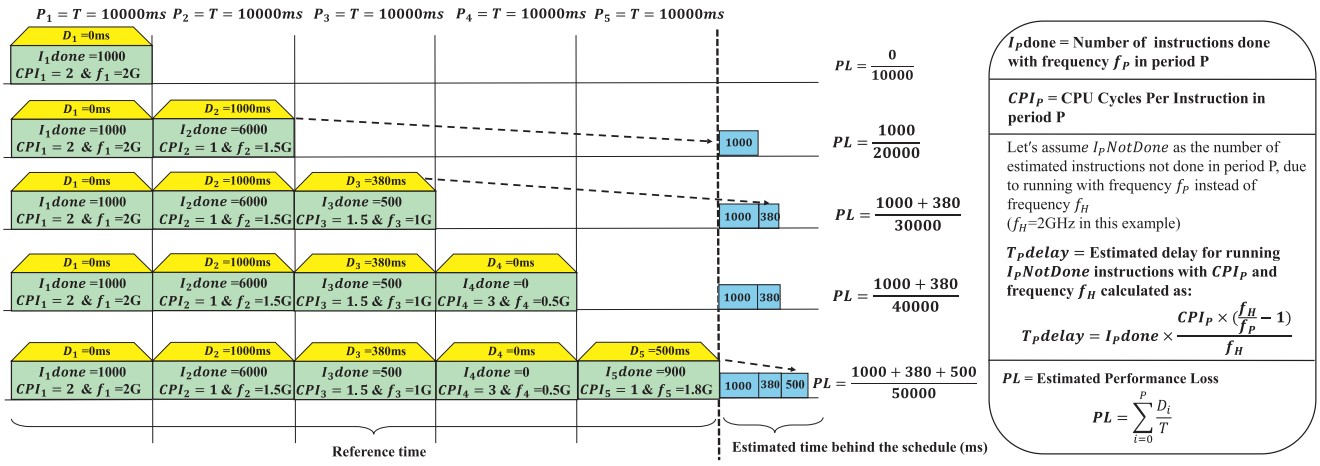


Fig. 3. Example utilized to illustrate the estimation of total performance loss (PL) so far, up to and including the currently completed control period and just before the start of a new control period for a given core.

Where N is the total number of periods and T is the duration or length of the control period.

Knowing previously selected frequencies for each of the cores of the CMP that were used in the last control period, together with the statistics about how many instructions have been executed by each core and what was the average *CPI* in the last control period (in system simulators as well as on current operating systems running on real multicore hardware, these data are readily available), we use Eq. (19) to estimate how much aggregated extra delay we have incurred up to the current control period. This is illustrated for a very simple example in Fig. 3, where we can see for example that at the end of the first control period the performance loss (PL) is zero because the execution during the first control period was done at the highest clock frequency $f_1 = 2$ GHz. However, at the end of the second control period, we have incurred a performance loss of 1000/20000 because a lower frequency of $f_2 = 1.5$ GHz was used (see second row in Fig. 3), and so on.

4.2. Block diagram and pseudocode of proposed algorithm

The expression in Eq. (19) gives us a good measure of the loss suffered in the control period that just finished execution. However, we would like to use it to estimate the performance loss during the next, incoming control period and based on that to be able to make an informed decision about what voltage-frequency (VF) pair to use that gives us maximum energy reduction within the limit of allowable performance degradation. The issue now however is that we need a way to predict what the actual workload will be in the next control period. In other words, at the end of the control period index P, we need a predictor for instruction counts and CPI of the next control period, that of index $P+1$. To do that we use a Kalman filtering based approach. We use a Kalman filtering based approach because we found it to be the best compromise between complexity of implementation, efficiency, and accuracy of prediction while considering a history of w past control periods.

The block diagram from Fig. 4 is essentially implemented as a control algorithm inside our customized Sniper based CMP system simulation framework. During a regular simulation of a given application or benchmark, for a given architecture of the CMP, information about the activity counters (i.e., number of instructions executed by each core and CPI) is fed to our algorithm. Our algorithm is *dynamic*, i.e., applied directly at runtime, and does not need any *static* application analysis or profiling that would be done in advance in order to identify improvement possibilities. The execution of the given application is done as a series of control periods. The

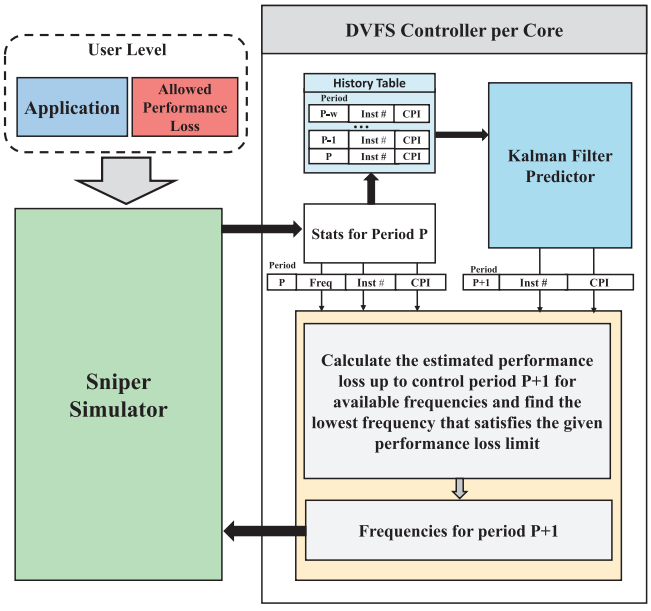


Fig. 4. Block diagram of the proposed DVFS based dynamic energy management (DEM) scheme as implemented inside our custom Sniper simulator.

information collected from the Sniper simulator at the end of each control period is recorded for a *moving window of w past periods* and is utilized to make predictions about the next control periods' instruction counts and CPI. The prediction is done with a Kalman filter based predictor as shown in Fig. 4.

The predictions are then used inside the algorithm for estimating the performance loss using Eq. (19) and for deciding the VF pairs for all cores for the next control period. The VF pairs are selected to maximize energy savings but without violation of the performance loss constraint, which is the user specified. We assume that, based on the criticality of the application, the user defines a tolerable performance loss ratio. The pseudocode of this control algorithm is shown in Algorithm 1 listing. The algorithm works with a list of VF pairs, out of which new VF pairs are selected for cores if that results into energy reduction without violating the performance degradation ratio threshold specified by the user. For each period, using the predicted instruction counts and cycles per instruction estimated by Kalman predictor for the next control period, the algorithm finds the lowest frequency to

Algorithm 1 Dynamic Energy Management (DEM) Under Performance Constraints.

Input:
α: acceptable performance loss ratio threshold; $I_{P_{done}}$, CPI_P for just ended control period
Output:
(V_{P+1}, f_{P+1}) VF pairs for all cores for next control period
Definitions:
T duration of each control period
$I_{(P+1)_{done}}$, CPI_{P+1} predicted with Kalman filter predictor
FreqList: list of available frequencies sorted from low to high (f_H)
$T_{delay} = 0$
$T_{ref} = 0$
if end of *control˜period* index P **then**
  for each *core* in CMP **do**

    $T_{ref} += T$
    $T_{delay} += \frac{I_{P_{done}} \times (\frac{f_H}{f_P} - 1) \times CPI_P}{f_H}$
    $Freq\_set = False$

    for $Freq$ in *FreqList* **do**

      $T_{ref_{next}} = T_{ref} + T$

      $PredT_{delay} = T_{delay} +$
      $\frac{I_{(P+1)_{done}} \times (\frac{f_H}{Freq} - 1) \times CPI_{P+1}}{f_H}$

      $PredPerfLoss_{P+1} = PredT_{delay} / T_{ref_{next}}$

      if $PredPerfLoss_{P+1} < \alpha$ **then**
        $f_{P+1} = Freq$
        $Freq\_set = True$
      end if
    end for
    if $Freq\_set = False$ **then**
      $f_{P+1} = f_H$
    end if
  end for
end if

save the maximum possible energy that satisfies the performance constraints.

5. Simulation results

We implemented the proposed dynamic energy management (DEM) scheme from Fig. 4 inside the Sniper based CMP system simulation framework [29], which is integrated with the McPAT power calculator that models all three main components of power, including dynamic, short-circuit, and leakage power [30]. We conducted extensive simulations on several Parsec and Splash2x benchmarks [31] to investigate the performance of the proposed algorithm. In our simulations, we used two different network-on-chip (NoC) based CMP architectures composed of 16 (4×4) cores and 64 (8×8) cores, respectively. Each of these architectures uses a regular mesh NoC topology. The default architectural configuration parameters utilized in our custom Sniper based simulations are listed in Table 1.

5.1. Accuracy of the Kalman filter predictor

As discussed earlier in the paper, our primary prediction technique uses Kalman filtering. This is used to predict the instruction count and the average CPI in the next control period for each core. Based on our simulations, we found empirically that a history window of 20 periods and values of 1, 1, 0.5, and 0.5 for *A, H, Q* and *R* filter parameters provided consistently very accurate predictions. Also it is assumed that there is no control input exists and *B* is set to 0. Therefore, we use these parameter values for all our simulations unless stated otherwise. For example, in Fig. 5, we show the predictions of both the CPI and the instruction count for a sample core while running the *radiosity* benchmark with 64 threads on a 64 core CMP architecture. The predicted values follow very closely the actual observed values, which turn out to occur at the end of the next control period. Therefore, we conclude that the Kalman filtering based prediction is very effective and computationally efficient. It serves well for our purpose, as it will be made clear in the next set of simulation results.

Table 1
Architectural configuration parameters.

Parameter	Value
Technology node	45 nm
Core	Intel X86
Core CPU model	Out of order (Detailed CPU)
Frequencies	2 GHz downto 1 GHz, with 100 MHz step
VDDs	$f \geq 1.8G$: 1.2*V*, $1.8G > f \geq 1.5G$: 1.1*V*, $1.5G > f \geq 1G$: 1*V*
Transition latency	2000 ns
Branch predictor	2 bit counter
Reorder buffer	80-entries
L1ICache/1core	32KB
L1DCache/1core	64KB
L2/1core	256KB
L3/4cores	8MB
Network	2D regular mesh, 1 router per core
Link bandwidth	64 bits

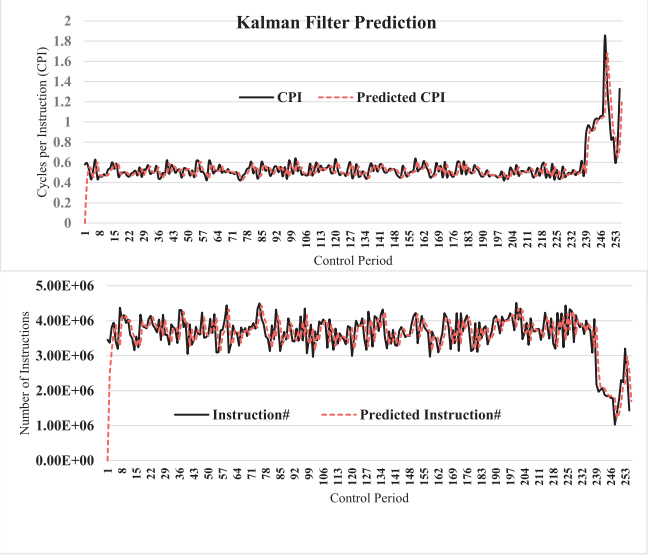
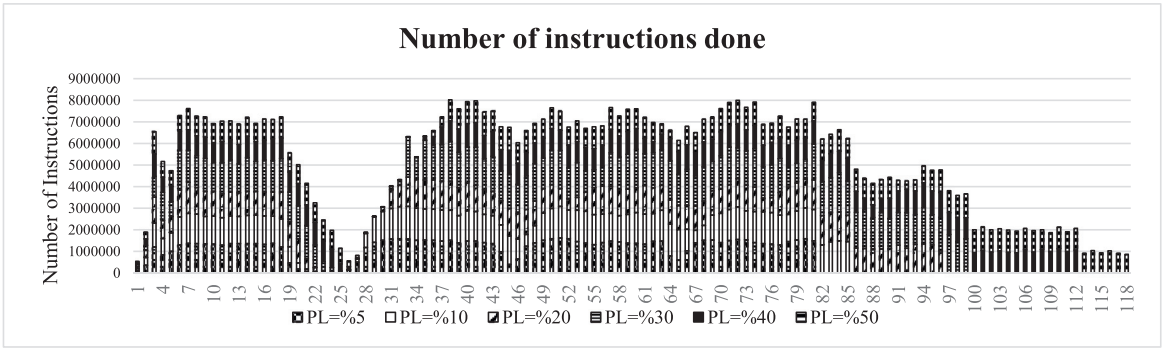


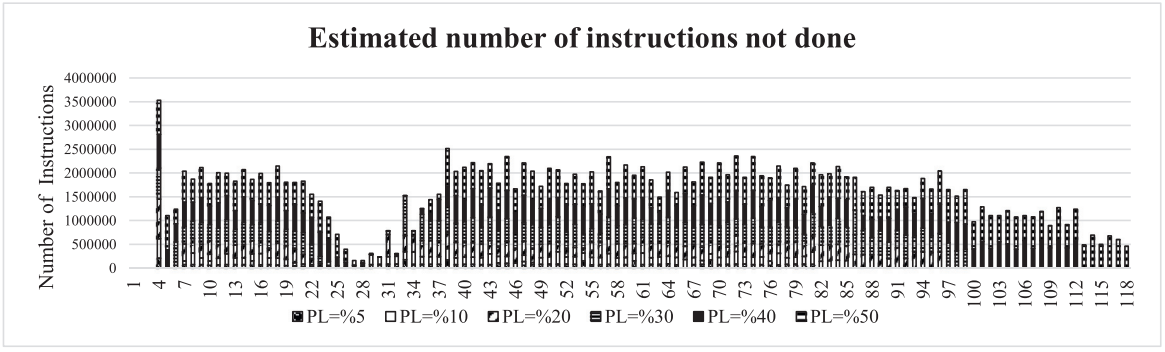
Fig. 5. Plots that show the comparison between the predicted values of the CPI and the instruction count for the next control period and the actual values that occurred and were observed at the end of the next control period. These traces are for a sample core (out of 64 cores) during the execution of *radiosity* benchmark.

5.2. Overhead due to Kalman filters

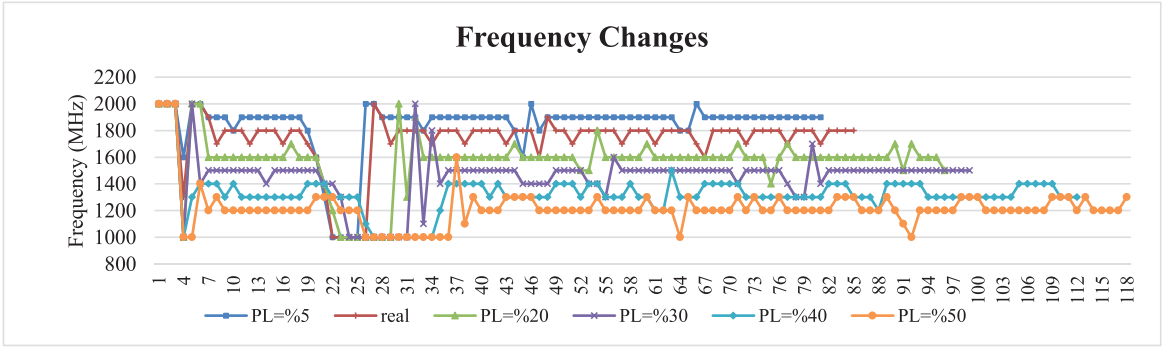
As mentioned earlier, the proposed dynamic energy management (DEM) uses a Kalman filter per core for scalability but most importantly for accuracy because we are interested in doing DVFS at per core level rather than for the whole processor. The Kalman



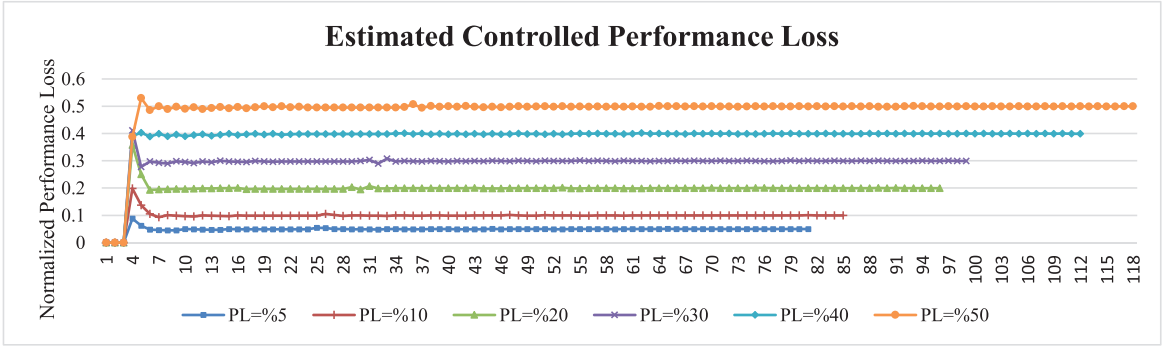
(a)



(b)



(c)



(d)

Fig. 6. Simulation results for a sample run of the *barnes* benchmark. The x axis represents the index of the control periods. Note that when the frequency is higher, the total execution time, measured as *walltime*, is shorter and therefore the total number of control periods is smaller.

filter is implemented as a C++ routine, which is integrated inside the Sniper system simulator that we use for our simulations. Whenever the Kalman filter needs to be executed, this routine is called and it executes very fast. Its runtime is very small at less than 0.05% of the duration of a control period. This performance overhead is included in the measurement of the total execution time for a given benchmark. Thus, the performance for a given benchmark when the proposed DEM is used includes also the overhead due to the execution of the Kalman filter routines.

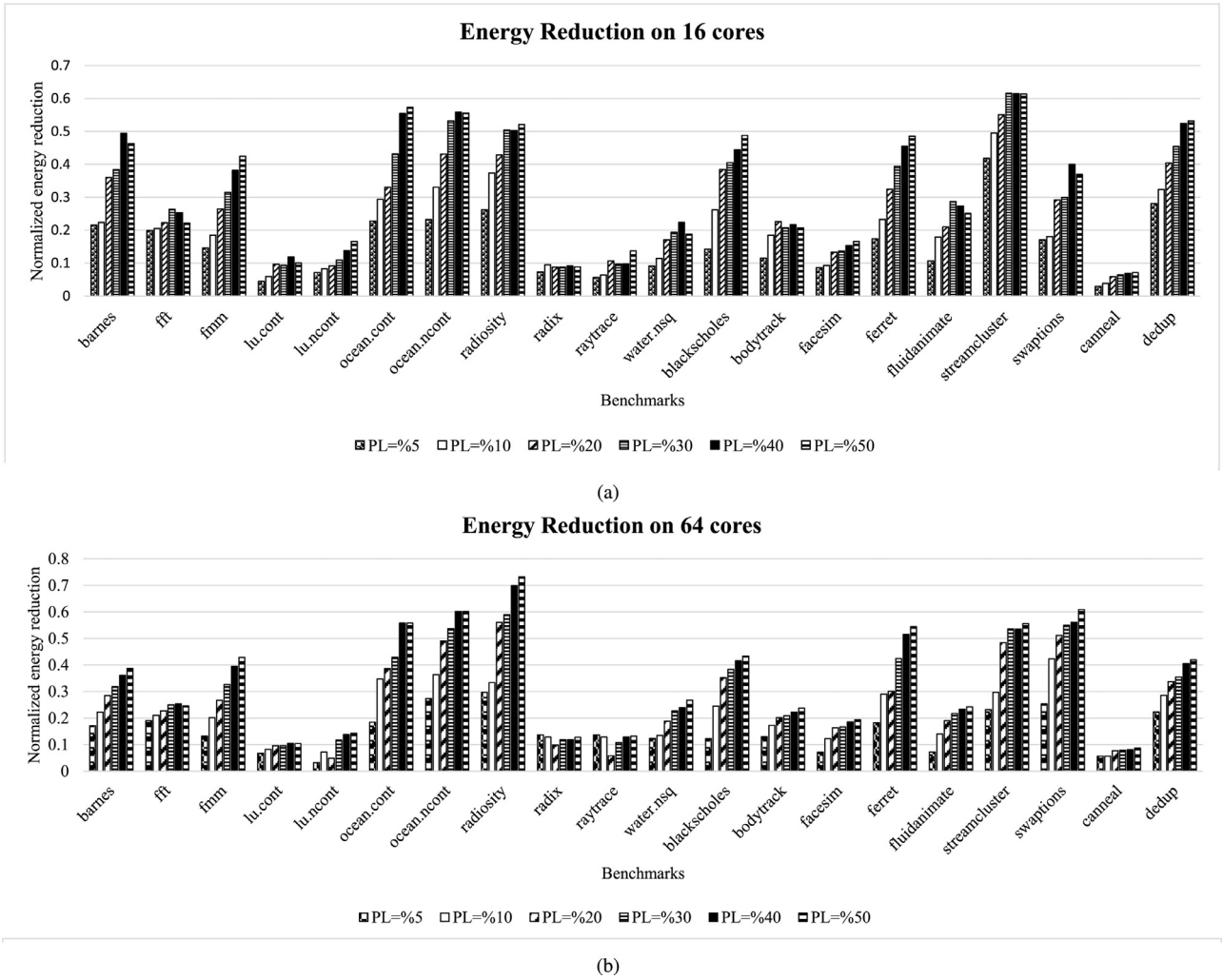


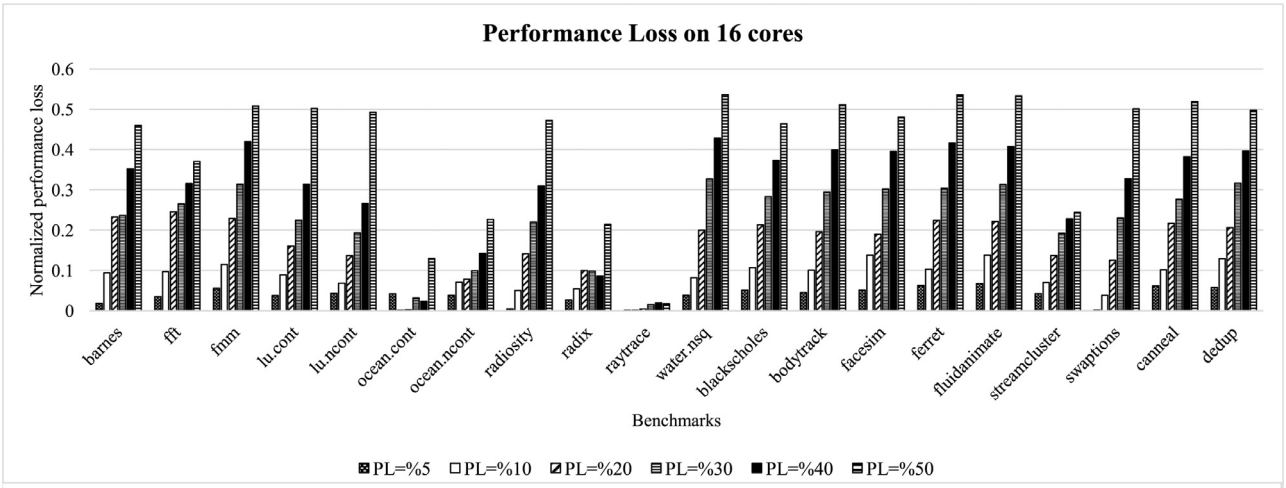
Fig. 7. Energy reduction percentages. (a) 16 core architecture with 4×4 mesh NoC. (b) 64 core architecture with 8×8 mesh NoC.

5.3. Dynamic energy management

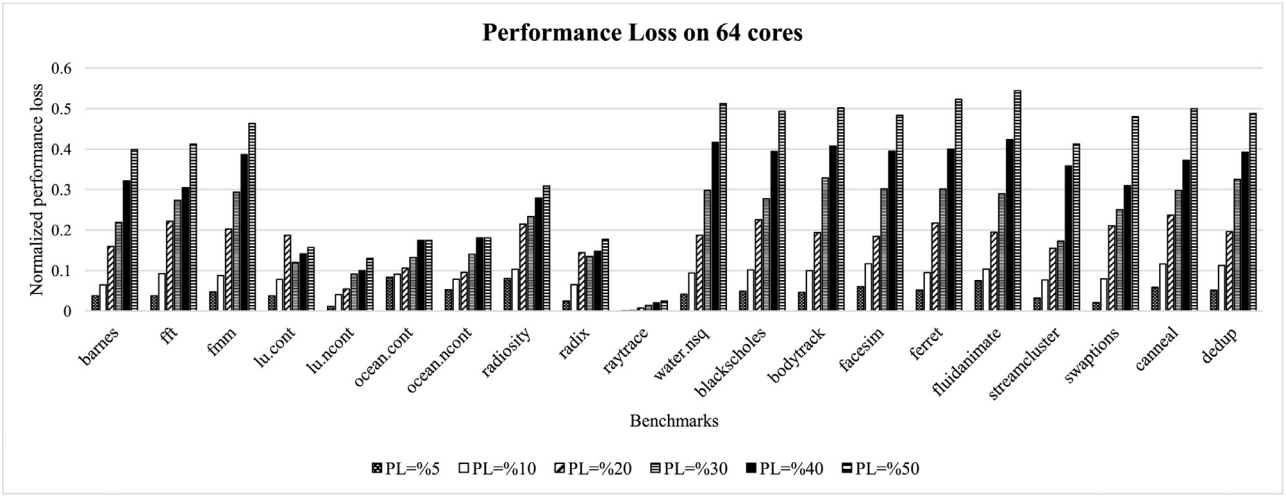
In the next set of simulation experiments, we investigate the performance of the proposed algorithm for several different values of the performance loss (PL) constraint or threshold. Recall that this constraint is set by the user who has the best knowledge about what is the acceptable performance degradation for a given application. We assume that the user sets this constraint based on her knowledge of the criticality of the application at hand. The higher the criticality, the lower the PL should be selected. We considered six different PL values including 5%, 10%, 20%, 30%, 40% and 50%. For example, a value of $PL = 5\%$ means that the user wants the proposed algorithm to try to reduce energy consumption as much as possible but without incurring a performance degradation of more than 5%.

Therefore, for each such PL value, we run the proposed DVFS based dynamic energy management (DEM) algorithm to find out what is the maximum achievable energy reduction under the specified performance degradation constraint. The results reported here focus on the so called region of interest (ROI) during the execution of a given benchmark. During each simulation, the custom Sniper simulator is stopped periodically after a constant amount of time (i.e., control period) and the proposed DEM algorithm described in Fig. 4 is called to find the best VF pairs for all cores for the next control period. In our simulations the control period is set to 1 ms, but it can be set to other values as well. During each such stop, the DEM algorithm estimates the delayed instructions count (DIC) in current control period and predicts the behaviour of the application on each core for the next period using Kalman predictor. Then, it tries to control the delay incurred due to the delayed instructions by selecting the lowest possible frequency, which still satisfies the acceptable performance penalty threshold.

The plots in Fig. 6 show simulation data collected during a sample run of the *barnes* benchmark on a 64 core architecture for the four different PL constraints. The plot in Fig. 6a shows the number of instructions executed during each control period on one of the 64 cores of the CMP architecture. Please contrast that with the predicted number of instructions that are delayed for later execution shown in Fig. 6b. This is the number of instructions that could have been executed in addition during the just completed control period, if the highest frequency had been used. In other words, because the execution in the just completed control period was done at a lower frequency (as dictated by the DVFS algorithm, in order to reduce energy consumption), these instructions are delayed and thus their execution will be "rolled over" during the incoming control periods. This plot is as we expected; the larger the threshold for performance loss (PL), the larger the number of instructions that are not completed and postponed for later. This in turn will result in longer overall execution time for a given benchmark.



(a)



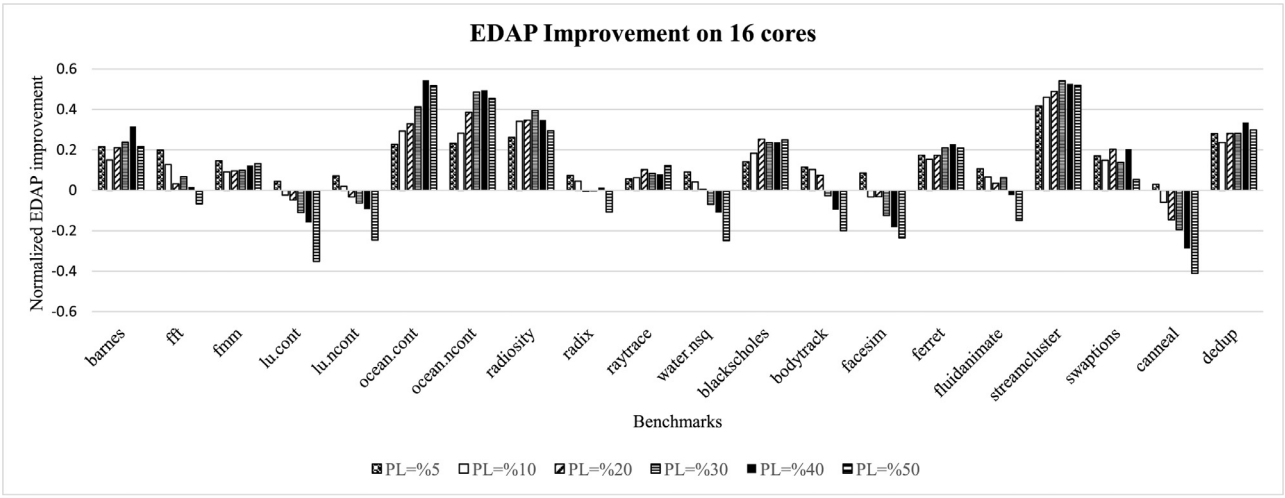
(b)

Fig. 8. Performance loss percentages. (a) 16 core architecture with 4x4 mesh NoC. (b) 64 core architecture with 8x8 mesh NoC.

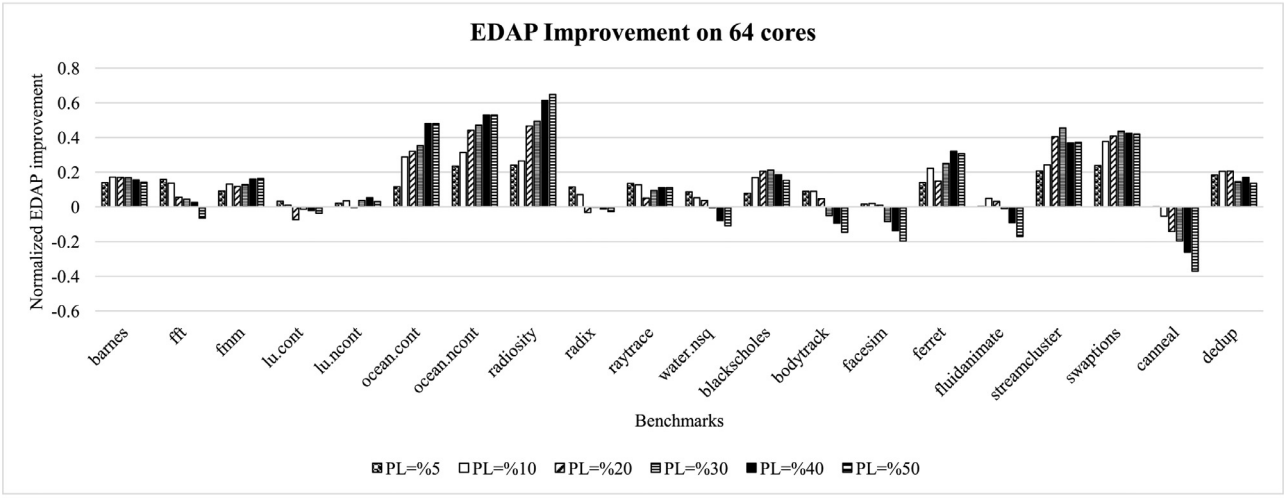
The way frequency was varied is shown in Fig. 6c while the calculated performance loss at the end of each control period is shown in Fig. 6d. We can see that for a tight PL threshold like 5%, the core frequency is higher than when the threshold is large. In other words, for example, when the PL threshold is relaxed to say 50%, the algorithm pushes the frequency way down in order to save as much energy as possible while trying to keep the estimated performance loss within the limit of 50%, as seen in the top curve of the plot in Fig. 6d. Similar plots can be collected for any of the 64 cores of the CMP architecture and for any of the simulated benchmarks. They are not reported here due to lack of space. Noteworthy, we observe that sometimes the estimated performance loss overshoots as shown by the curve corresponding to the PL threshold of 50% in Fig. 6d. This happens when the frequency for the just completed control period was selected too low. As a result the performance degradation violates the desired threshold for a short period of time. This is a direct result of the prediction error that was experienced at the end of the previous control periods. We do not have currently a way to eliminate these “artifacts”, unless we wanted to become over-conservative in the way we allow the proposed DVFS algorithm to throttle core frequencies. Therefore, we consider these short lived PL threshold hikes as acceptable.

The energy reduction for all the benchmarks that we investigated is shown in Fig. 7 for two different CMP architectures. The y axis of these plots shows normalized values for simplicity and clarity. For example, a value of 0.2 on the y axis of Fig. 7a means a 20% energy reduction. We can see that energy savings are consistent across the board and, as expected, the savings are bigger when the performance loss constraint is more relaxed. Note that for some benchmarks the energy savings can be as high as 60% for either of the two CMP architectures. The performance loss for all benchmarks and for both CMP architectures is shown in Fig. 8. Please note that the plots in this figure actual performance loss as calculated and reported by the Sniper tool simulator, and it includes also the overhead of executing the Kalman filter routines. Again, the y axis shows normalized values similar to the plots in Fig. 7. Note that for each of the four different values for the PL threshold, the actual performance loss calculated at the end of the execution of each benchmarks is kept within limits reasonably well. Some benchmarks experience slightly larger performance degradation and that is due to the “artifacts” discussed earlier.

Finally, we also show in Fig. 9 the change in energy delay area product (EDAP). In all cases the area is actually the same and does not really affect these plots. We can see that in the majority of the cases the EDAP is improved. In some difficult instances that is not



(a)



(b)

Fig. 9. Energy Delay Area Product (EDAP) percentages. (a) 16 core architecture with 4×4 mesh NoC. (b) 64 core architecture with 8×8 mesh NoC.

the case. The reason for that is because we apply the proposed algorithm and report results only for the so called region of interest (ROI) of a given benchmark and not for the whole duration of execution. These benchmarks are created such that during the ROI all cores are fully utilized and thus usually there is little room for improving performance via frequency theottling when everything is busy almost all the time. In experimental setups like this, it is unlikely that both energy consumption and performance can be improved because all cores are working all the time. This is not so in some previous works, which reported wishfull energy consumption reduction and performance improvement at the same time. We suspect that is possible only if simulations are monitored outside the ROI where some of the cores do not have threads scheduled and thus one can find room for execution optimization. This is something that is actually unclear in previous works, which do not discuss whether their results are reported for ROIs or not.

5.4. Qualitative comparison with previous work

We do not report direct comparisons of the proposed algorithm with other previously proposed dynamic energy management (DEM) solutions. That is mainly because the implementation of previous solutions is not available. In addition, previous solutions were for single or relatively small multicore processor architectures. The number of VF settings vary widely among different studies. Their architectures used bus based instead of NoC based communication in many instances. Simulations were reported for different benchmarks whose parallelism or inputs are many times not specified or discussed. Other studies reported experiments done on real hardware on processors and operating systems that we may not have in our setup. Some previous studies report different figures of merit such as MIPS/Watt or energy per user-instruction, which are different from direct execution time or actual power numbers. This makes replication of results very difficult if not impossible, especially if they were reported for processor architectures different than ours. Moreover, we see as our main contribution the investigation of the tradeoff between achievable energy reduction for a given performance penalty threshold, investigation which we do using a novel model, efficient and accurate DVFS algorithm.

Nevertheless, we include a qualitative comparison with some of the recently reported DVFS algorithms. Please note that this is a qualitative comparison in order to get an idea about the capability of the solution as a DVFS strategy. This qualitative comparison is presented in Table 2. We can see that the proposed DVFS algorithm is comparable with previous solutions. However, the proposed approach uses a Kalman filter based technique that is efficient and accurate while the DVFS algorithm has the capability of tuning the

Table 2
Qualitative comparison to previous work.

Approach	Avg. energy reduction (%)	Avg. performance degradation (%)
Proposed (16 cores, Sniper)	15,20,26,30,33,34	5,10,20,30,40,50
Proposed (64 cores, Sniper)	15,22,27,30,34,35	5,10,20,30,40,50
[10] (singlecore Intel PXA27x)	34	17
[12] (quad-core processor)	28	negligible
[18] (dual-core AMD Opteron 2218 nodes)	20	4
[21] (Intel i5-4590 quad-core nodes)	15	4.8
[22] (dual-core Intel core 2 nodes)	22	20
[15] (2 SPARC-like cores and)	12	maintains
4 XScale-like cores)		target perf.
[23] (Superscalar processor)	20	10
[32] (16 cores, Sniper)	5,5,5	5,10,15
(8 cores, Sniper)	8,7.5,7	5,10,15
(4 cores, Sniper)	2,3,4	5,10,15
[33] (dual core processor)	41,43,45	30,59,99

tradeoff between energy reduction and performance degradation as dictated by the user.

6. Conclusion

We introduced a novel algorithm for dynamic energy management under performance constraints in chip multi-processors. It uses Kalman filtering to predict instruction counts and average cycles per instruction for the incoming control periods for all cores of the CMP. These predictions are then used as the basis on which voltage-frequency pairs are selected for each core by a novel dynamic voltage and frequency scaling algorithm whose objective is to reduce energy consumption but without degrading performance beyond the user set threshold. Simulations results on network-on-chip based CMP architectures with 16 and 64 cores demonstrated the effectiveness of the proposed algorithm on the vast majority of the simulated benchmarks. The complete implementation of our algorithm and scripts for duplicating the results will be available for download on our website.

As future work, it would be interesting to study how the proposed model and algorithm could be improved by considering also the DRAM memory, which was shown to have a significant impact on the overall power [21] as well as the amount of memory or I/O access (during which the cores may need to stall) which needs to be considered because the ratio of the off-chip and on-chip execution is also critical in determining the performance loss as shown in [22].

Acknowledgment

This work was supported by the Dept. of Electrical and Computer Engineering at Marquette University.

References

[1] J. Whitney, P. Delforge, Data Center Efficiency Assessment - Scaling Up Energy Efficiency Across the Data Center Industry: Evaluating Key Drivers and Barriers, Natural Resources Defense Council (NRDC) Report, 2014. [Online]. Available: https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf.

[2] Annual energy outlook u. s. energy information administration (EIA), 2016, [Online]. Available: http://www.eia.gov/forecasts/aeo/data.cfm#enconsec.

[3] , United States Environmental Protection Agency, Report to Congress on Server and Data Center Energy Efficiency, Report, 2007. [Online]. Available: https://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf.

[4] W. Kim, M.S. Gupta, G.-Y. Wei, D. Brooks, System level analysis of fast, per–core DVFS using on-chip switching regulators, IEEE Int. Symposium on High Performance Computer Architecture (HPCA), 2008.

[5] T. Kolpe, A. Zhai, S.S. Sapatnekar, Enabling improved power management in multicore processors through clustered DVFS, in: ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE), 2011.

[6] K. Chakraborty, S. Roy, Architecturally homogeneous power-performance heterogeneous multicore systems, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 21 (4) (2013) 670–679.

[7] A.A. Sinkar, H.R. Ghasemi, M.J. Schulte, U.R. Karpuzcu, N.S. Kim, Low-cost per–core voltage domain support for power-constrained high-performance processors, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 22 (4) (2014) 747–758.

[8] R. Jevtic, H.-P. Le, M. Blagojevic, S. Bailey, K. Asanovic, E. Alon, B. Nikolic, Per–core DVFS with switched-capacitor converters for energy efficiency in many-core processors, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 23 (4) (2015) 723–730.

[9] R. Schone, T. Ilsche, M. Bielert, D. Molka, D. Hackenberg, Software controlled clock modulation for energy efficiency optimization on intel processors, IEEE Int. Workshop on Energy Efficient Supercomputing (E2SC), 2016.

[10] G. Dhiman, T.S. Rosing, Dynamic voltage frequency scaling for multi-tasking systems using online learning, ACM/IEEE Int. Symposium on Low Power Electronics and Design (ISLPED), 2007.

[11] M. Moeng, R. Melhem, Applying statistical machine learning to multicore voltage and frequency scaling, in: ACM Int. Conference on Computing Frontiers, 2010.

[12] H. Jung, M. Pedram, Improving the efficiency of power management techniques by using bayesian classification, Int. Symposium on Quality Electronic Design (ISQED), 2008.

[13] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, L. Torres, Dynamic and distributed frequency assignment for energy and latency constrained MP-soc, in: ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE), 2009.

[14] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, G.D. Micheli, Temperature control of high-performance multi-core platforms using convex optimization, in: ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE), 2008.

[15] S. Sharifi, A.K. Coskun, T.S. Rosing, Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs, in: ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), 2010.

[16] V. Hanumaiah, D. Desai, B. Gaudette, C.J. Wu, S. Vrudhula, STEAM: a smart temperature and energy aware multicore controller, ACM Trans. Embedded Comput. Syst. (TECS) 15 (13) (2014).

[17] C.H. Hsu, W.-C. Feng, A power-aware run-time system for high-performance computing, in: ACM/IEEE Conference on Supercomputing, 2005.

[18] R. Ge, X. Feng, W. Feng, K.W. Cameron, CPU MISER: a performance-directed, run-time system for power-aware clusters, IEEE Int. Conf. Parallel Process. (ICPP) (2007).

[19] S. Huang, W. Feng, Energy-efficient cluster computing via accurate workload characterization, IEEE/ACM Int. Symposium on Cluster Computing and the Grid (CCGRID), 2009.

[20] B. Rountree, D.K. Lownenthal, B.R. de Supinski, M. Schulz, V.W. Freeh, T. Bletsch, Adagio: making DVS practical for complex HPC applications, in: ACM/IEEE Conference on Supercomputing, 2009.

[21] V. Sundriyal, M. Sosonkina, Joint frequency scaling of processor and DRAM, J. Supercomput. 72 (4) (1549) 154–1569.

[22] V. Sundriyal, M. Sosonkina, F. Liu, M.W. Schmidt, Dynamic frequency scaling and energy saving in quantum chemistry applications, IEEE Int. Parallel and Distributed Processing Symposium (IPDPS), 2011.

[23] G. Keramidas, V. Spiliopoulos, S. Kaxiras, Interval-based models for run-time DVFS orchestration in superscalar processors, in: ACM Int. Conference on Computing Frontiers, 2010.

[24] B. Rountree, D.K. Lowenthal, M. Schulz, B.R. de Supinski, Practical performance prediction under dynamic voltage frequency scaling, in: Int. Green Computing Conference and Workshops, 2011.

[25] R. Miftakhutdinov, E. Ebrahimi, Y.N. Patt, Predicting performance impact of DVFS for realistic memory systems, Int. Symposium on Microarchitecture (MICRO), 2012.

[26] S. Eyerman, L. Eeckhout, A counter architecture for online DVFS profitability estimation, IEEE Trans. Comput. 59 (11) (2010) 1576–1583.
[27] G. Welch, G. Bishop, An Introduction to the Kalman Filter, Univ. North Carolina, Chapel Hill, Chapel Hill, NC, 1995.
[28] S. Bang, K. Bang, S. Yoon, E. Chung, Run-time adaptive workload estimation for dynamic voltage scaling, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD) 28 (9) (2009) 1334–1347.
[29] T.E. Carlson, W. Heirman, L. Eeckhout, Sniper: exploring the level of abstraction for ssalable and accurate parallel multi-core simulation, in: Int. Conference for High Performance Computing, Networking, Storage and Analysis, 2011.
[30] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, N.P. Jouppi, McPAT: an integrated power, area, timing modeling framework for multicore and many-core architectures, Int. Symposium on Microarchitecture (MICRO), 2009.
[31] PARSEC and splash2 benchmarks, 2017,. [Online]. Available: http://parsec.cs.princeton.edu.
[32] W. Lee, Y. Wang, M. Pedram, VRCon: dynamic reconfiguration of voltage regulators in a multicore platform, in: Proceedings of the conference on Design, Automation & Test in Europe (DATE), 2014.
[33] H. Shen, L. Jun, Q. Qinru, Learning based DVFS for simultaneous temperature, performance and energy management, IEEE Int. Symposium on Quality Electronic Design (ISQED), 2012.

Milad Ghorbani Moghaddam received the B.S. degree from Ferdowsi University of Mashhad, Iran in 2008 and the M.Sc. degree from Isfahan University of Technology, Iran in 2011, both in computer engineering. Currently, he is a Ph.D. student in the Department of Electrical and Computer Engineering, Marquette University, Milwaukee, WI, USA. His main research interests include lifetime reliability of chip multiprocessors and full system simulators.

Cristinel Ababei received the Ph.D. degree in electrical and computer engineering from the Univ. of Minnesota, Minneapolis, in 2004. He is an assistant professor in the Dept. of ECE, Marquette Univ. Prior to that, from 2012 to 2013, he was an assistant professor in the Dept. of EE, SUNY at Buffalo. Between 2008 to 2012, he was an assistant professor in the Dept. of ECE, North Dakota State University. From 2004 to 2008, he worked for Magma Design Automation, Silicon Valley. His current research interests include design automation of systems-on-chip with emphasis on reliability, FPGAs, and parallel computing.